ASYNCHRONOUS CHANGE CAPTURE FOR DATA WAREHOUSING

RELATED APPLICATIONS

IN. A1

[01] The present application is related to U.S. Patent Application Serial No. 10/140,818 entitled "Method and Apparatus for Change Data Capture in a Database System" filed on _____ (attorney docket no. 50277-1002, client docket no. 2000-190-01) by William D. Norcott et al., the contents of which are hereby incorporated by reference.

[02] The present application is related to U.S. Patent Application Serial No. 09/863,412 entitled "Synchronous Change Data Capture in a Database System" filed on _____ (attorney docket no. 50277-1006, client docket no. 2001-010-01) by William D. Norcott, the contents of which are hereby incorporated by reference.

FIELD OF THE INVENTION

[03] The present invention relates to database systems and more particularly to asynchronous change capture for data warehousing.

BACKGROUND OF THE INVENTION

[04] Many businesses and other large organizations today use relational database management systems known as on-line transaction processing (OLTP) systems to execute and keep track of business transactions. For example, a company that sells products or services over the Internet may use an OLTP system to record pricing information about each product for sale, billing and shipping information for each purchaser, and sales information for each order made by a purchaser. Other examples of businesses that use OLTP systems include airlines, banks, mail-order companies, supermarkets, and manufacturers.

[05] The data generated and recorded in OLTP systems are valuable to most businesses, because the businesses can aggregate and analyze the data to ascertain the product sales for a particular month, forecast changing trends in product popularity and identify profitable or

unprofitable product lines, or otherwise evaluate the businesses' affairs. Aggregating and analyzing this data, however, is computationally expensive and, if performed on the OLTP system itself, would decrease the performance of the OLTP system. Accordingly, it has become common for businesses with OLTP systems to set up a separate computer system, generally known as a "data warehouse," for the purpose of collecting, aggregating, and analyzing the information contained in the OLTP databases. Data warehouses can grow very large, ranging from gigabytes to many terabytes of data (trillions of bytes). The task of moving data from its original source in OLTP systems to the data warehouse is commonly referred to as data extraction, transport, and loading (ETL).

[06] Conventional data extraction, transport, and loading mechanisms are cumbersome. In a typical approach, database administrators generally dump the entire contents of the tables in the OLTP system into flat files, transport the flat files to a staging system, and then load the data in the flat files into the data warehouse. In this approach, the amount of data extracted, transported, and loaded is as immense as the amount of data in the OLTP system, even though only a small percentage of the data on the OLTP system is actually new. Accordingly, there has been much interest in devising ways to reduce the amount of data extracted, transported, and loaded by capturing only the changed data to the database tables of the OLTP system.

[07] A typical approach for capturing the changed data of OLTP system database tables is to add a column to the OLTP system database tables to store a timestamp or a sequence number and conditionally extract the data that has the newest timestamps or sequence numbers. Thus approach has several disadvantages. First, this approach requires a change to the schema, e.g. adding the extra column to hold the timestamp to track the changes. Not only is this schema change an administrative nightmare, but many vendors of OLTP systems forbid their customers from making any changes to the schema for security, maintenance, and liability reasons.

IDS. A2

[08] Second, there is a performance penalty in storing the timestamp for every row of new or changed data, yet performance is critical for OLTP systems. Third, while timestamps can easily identify which rows have changed by due to an insert or update, timestamps cannot distinguish between a newly inserted row or an old row that was updated. Furthermore, timestamps cannot identify deleted rows, because deleted rows are no longer present in the database. The lack of

this kind of change information makes it difficult to properly update summaries of the OLTP data, resulting in expensive recalculations of the summary data.

[09] Another approach to capturing changed data is known as "synchronous change data capture," in which the changes are captured in the very same transaction that is updating the tables on the OLTP system. Thus, as new data arrives in the data warehouse, the changes made to one or more tables on the OLTP system are captured synchronously and stored in corresponding change tables in the data warehouse, such that for every table that is updated on the OLTP system, there is a corresponding change table that contains those changes.

[10] Conventional systems have used triggers for synchronous change data capture, either by using the CREATE TRIGGER statement or by using an internal mechanism with equivalent functionality. A trigger is an object that specifies a series of actions to be automatically performed when a specific event occurs, and, according to industry standards, the events that cause triggers to be activated (or "fired") are DML statements. For synchronous change data capture, triggers have been designed to fire when a row of a database table is updated, inserted, or deleted. Each of these modifications is associated with its own system change number (SCN), which is recorded by the trigger.

[11] Trigger-based change data capture has two disadvantages for many data warehousing deployments. The first disadvantage is performance. Because triggers are fired every time a row is updated, inserted, or deleted on the OLTP system, triggers impose a performance penalty on every operation performed on the OLTP system. The overhead for trigger processing can be substantial, requiring as much as three times the amount of computing resources to process the same number of operations without triggers. The second disadvantage is that the creation of a trigger is still technically a schema change and therefore forbidden by many turn-key OLTP systems.

[12] Therefore, there is a need for a high-performance technique of data extraction from OLTP systems. There is also need for a data extraction mechanism that does not require schema changes, either in source tables of the OLTP system or by creating triggers.

SUMMARY OF THE INVENTION

[13] These and other needs are addressed by the present invention by extracting the change data from the recovery logs generated by the OLTP system. During the normal course of operation, database management systems maintain a recovery log to allow users to undo transactions and to provide for recovery after a system crash. These logs can be shipped to a data warehouse on a separate computer system, where the change data stored in the recovery logs can be extracted without affecting the performance of the OLTP system.

[14] Accordingly, one aspect of the present invention involves a method and software for change data capture, in which change data is extracted from a recovery logand stored in a database object such as a change table. The change data stored in the database object indicates what modification has been performed to a source object on the OLTP system. In various embodiments of the present invention, a database statement may be generated and executed to extract and load the change data. The recovery log itself may be shipped from an OLTP system to a staging system.

[15] Still other aspects, features, and advantages of the present invention are readily apparent from the following detailed description, simply by illustrating a number of particular embodiments and implementations, including the best mode contemplated for carrying out the present invention. The present invention is also capable of other and different embodiments, and its several details can be modified in various obvious respects, all without departing from the spirit and scope of the present invention. Accordingly, the drawing and description are to be regarded as illustrative in nature, and not as restrictive.

BRIEF DESCRIPTION OF THE DRAWINGS

[16] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[17] FIG. 1 is a high-level architectural diagram of one embodiment of the present invention.

[18] FIG. 2 is a schematic diagram depicting change objects in accordance with one embodiment of the present invention.

[19] FIG. 3 is a flowchart that illustrates asynchronous change data capture for an embodiment of the present invention.

[20] FIG. 4 depicts a computer system that can be used to implement an embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

[21] A system, method, and software for change data capture are described. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It is apparent, however, to one skilled in the art that the present invention may be practiced without these specific details or with an equivalent arrangement. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

[22] In a database management system, data is stored in one or more data containers, each container contains records, and the data within each record is organized into one or more fields. In relational database systems, the data containers are referred to as tables, the records are referred to as rows, and the fields are referred to as columns. In object-oriented databases, the data containers are referred to as object classes, the records are referred to as objects, and the fields are referred to as attributes. Other database architectures may use other terminology.

[23] Systems that implement the present invention are not limited to any particular type of data container or database architecture. However, for the purpose of explanation, the terminology and examples used herein shall be that typically associated with relational databases. Thus, the terms "table," "row," and "column" shall be used herein to refer respectively to the data container, record, and field.

ARCHITECTURAL OVERVIEW

[24] FIG. 1 depicts the architecture of one embodiment of the present invention comprising a source system 110 and a staging system 120. The source system 110 is typically an on-line transaction processing (OLTP) for executing and keeping track of transactions for a business. For example, the source system 110 hosts a business application 111 that is responsible for interacting with employees or customers of the business. In response to commands and queries from the user of the business application 111, the business application 111 interacts with an OLTP database 113 for storing and retrieving data.

[25] Functioning as the data warehouse in this example, the staging system 120 hosts one or more subscriber applications 121, 123. Without loss of generality, two subscriber applications 121, 123 are depicted, although any number may be created during the course of operation of an embodiment of the present invention. The subscriber applications 121, 123 are responsible for aggregating and analyzing the change data that has been extracted from the OLTP database 113, transported to the staging system 120, and loaded into the analysis database 125. Preferably, one of two mechanisms are employed to extract data from the OLTP database 113 without using flat files. These mechanisms are termed "synchronous extraction" and "asynchronous extraction," respectively.

INS. A5

[26] In the synchronous extraction mechanism, triggers 115 are employed to capture each change to the OLTP database 113 when the changes are made and transport the changes to the staging system 120. At the staging system 120, these changes are then integrated and loaded into change tables (not shown) of the analysis database 125 by a publisher process 127. The synchronous extraction mechanism is described in greater detail in the commonly assigned, co-pending U.S. Patent Application Serial No. _____ entitled "Synchronous Change Data Capture in a Database System" filed on _____ (attorney docket no. 50277-1006, client docket no. 2001-010-01) by William D. Norcott, the contents of which are hereby incorporated by reference.

[27] For the asynchronous extraction mechanism, which is described in greater detail herein below, a log shipper 119 periodically copies recovery logs 117 that are produced by the OLTP database 113 in the normal course of operation. The recovery logs 117 contain all the changes that have been applied to the OLTP database 113 and are used for backing up the data in the OLTP database 113 and restoring the data in case of a system crash. The log shipper 119 copies the recovery logs 117 to an area of the staging system 120 called a change source 131, which can be implemented as an operating system directory. The publisher 127 interacts with a log viewer process 129 to obtain the change data from the shipped recovery logs in the change source 129 without having to be aware of the internal implementation details of the recovery logs. The publisher 127 then loads the change data obtained via the log viewer process 129 into the change tables in the analysis database 125 for use by the subscriber applications 121, 123. The interaction of the subscriber applications 121, 123 with the change tables is described in the

commonly assigned, co-pending U.S. Patent Application Serial No. _____ entitled "Method and Apparatus for Change Data Capture" filed on _____ (attorney docket no. 50277-1002, client docket no. 2000-190-01) by William D. Norcott et al., the contents of which are hereby incorporated by reference.

OBJECTS FOR MANAGING CHANGE DATA

[28] In accordance with one aspect of the present invention, the change data extracted from the OLTP database 113 is maintained not in a flat file but in one or more database objects, referred to herein as "change tables" under control of a database management system, e.g. analysis database 123. Because the database management system provides such features as crash recovery for security and indexing for performance, use of database objects to hold the change data advantageously attains these beneficial features, without additional programming as compared to use of flat files.

[29] Referring to FIG. 2 by way of example, each source table or database object on the OLTP database 113 that is subject to change data capture is associated with a corresponding change table 211, 213, 221, 223, 225 in the analysis database 123. For transactional consistency, change tables 211, 213, 221, 223, 225 are grouped into sets of one or more "change sets " 210, 220 such that the publisher 125 ensures that all new change data added to the change tables in the same change set (e.g. changes tables 211, 213 of change set 210) are added at the same time, e.g. the modifications to these changes tables are performed in the same transaction and committed. In the example depicted in FIG. 2, there are two change sets, change set 210 and change set 220. Change set 210 comprises change table 211 and change table 213, which correspond to respective tables (not shown) on the OLTP database 113. Likewise, change set 220 comprises change table 221, change table 223, and change table 225, which also correspond to respective tables (not shown) on the OLTP database 113. The information that defines the structure of the change sets 210, 220 and change tables 211, 213, 221, 223, 225 is maintained in system metadata 230.

[30] Each change table employs a number of control columns in addition to the source table columns whose values were extracted, transported, and loaded from columns of the

corresponding source table in the OLTP database 113. In the example of FIG. 2, change table 225 is depicted as having a set of source table columns 231 and control columns SCN 233, TIME 235, OP 237, and ROW 239. The source table columns 231 may include all or a subset of the columns in the corresponding source table. In various implementations, the control columns may be part of the same database object that contains the source table columns or part of parallel, associated database object, which can be joined with source table columns (e.g. by a row identifier or a primary key).

[31] The control columns SCN 233, TIME 235, OP 237, and ROW 239 preferably have reserved names that customers are not allowed to use for their own columns, for example, names with a reserved character such as a dollar sign ($). The reserved names, of course, can be any valid string and, in this example, are named SCN 233, TIME 235, OP 237, and ROW 239 for mnemonic reasons.

[32] The SCN 233 column holds the System Change Number of the commit for the transaction on the OLTP database 113 that gave rise to the change data. A system change number is a monotonically increasing number that identifies every operation performed on a database system, e.g. update, insert, delete, and commit, that can be used to order the operations performed in the database system. The present invention is not limited to any particular implement of system change numbers, and the concepts disclosed herein may be profitably employed with timestamps, incrementing serial numbers, and the like.

[33] The TIME 235 column contains the commit time of the transaction that gave rise to the change data. This column helps subscriber applications 121 select or view change data that occurs in particular periods of time.

[34] The OP 237 column contains a code indicating the type of operation that resulted in the change data. For example, if the OP 237 column contains the symbol 'I', then the operation was an insert operation and the change data in the source table columns 231 contains the data that was inserted. If the OP 237 column contains the symbol 'D', then the operation was a delete operation and the change data in the source table columns 231 contains the data that was deleted (this allows the summaries to be properly readjusted by the subscriber applications 121). If the OP 237 column contains the symbol 'UO', then the operation was an update operation and the

change data in the source table columns 231 contains the old data that was modified; likewise, if the OP 237 column contains the symbol 'UN', then the operation was an update operation and the change data in the source table columns 231 contains the new data that was modified. Thus, update operations on the source table result in two entries in the change table, one for the old data and another for the new data, so the subscriber applications 121 have sufficient information to readjust their summaries of the OLTP data. Under certain circumstances, the source table may include "large objects" (LOB) such as graphics and pictures. This LOB is typically not summarized, so, to save memory, only the new LOB is recorded in the change table (with the symbol 'UL' in the OP 237 column). These symbols in the OP 237 column ('I', 'D', 'UO', 'UN', 'UL') are chosen for mnemonic reasons and other symbols can be arbitrarily assigned any unique value without departing from the purview of the present invention (e.g. numbers).

[35] The ROW 239 column contains a unique row sequence number for the changes in the change table. This column helps subscriber applications 121 order the operations that have been committed in the same transaction. Also, both the record for the old updated values (OP 237 column with 'UO') and the record for the new updated values (OP 237 column with 'UN') of the same operation will have the same row sequence number, because these two records are for the same change operation on the OLTP database 113 (an update operation).

INS. A7

[36] Although not depicted in FIG. 2, additional control columns may be provided to facilitate the implementation of embodiments of the present invention. For example, a bit mask of the updated columns can be used to identify quickly which columns have changed. As another example, the name of user who cause the operation can be recorded in a control column. The row identifier of the affected row in the source table can also be included in a control column.

[37] The subscriber applications 121, 123 of FIG. 1, however, need not see all the contents of the change tables. In particular, the range of rows that the subscriber application 121, 123 sees in the respective subscriber view is restricted and carefully controlled, as explained in detail below, so that change data is not lost nor double counted for subscriber applications 121, 123. In the example of FIG. 2, two subscriber views 241, 243 are depicted although any number of subscriber views may be created during the operation of an embodiment of the present invention. Use of the subscriber views 241, 243 beneficially insulates the respective subscriber applications

121, 123 from the implementation details of the change table 225. Unlike some prior art approaches, it is not necessary to add any control columns or information to the source tables themselves on the OLTP database 113; the provision of control columns for the change tables on the analysis 120 suffices. Consequently, this feature allows change data capture to be performed without changing the schema of the OLTP database 113, which is desirable for many turn-key OLTP databases.

ASYNCHRONOUS CHANGE DATA CAPTURE

[38] Referring to FIG. 3, at step 310, the recovery logs 117 are shipped from the source system 110 to a location on the staging system 120 called a change source 131, which can be an operating system directory. Also known as a "redo log," a recovery log 117 is produced by the OLTP database system 113 in the normal course of operation to allow users to undo transactions (e.g. in a transaction rollback) and to provide for recovery after a system crash. Recovery logs 117 thus provide a way to cancel to abort a transaction before the transaction is committed, and recovery logs 117 are a standard feature of relational database management systems. One way to implement the log shipper 119 is described in the commonly assigned, co-pending U.S. Patent Application Serial No. _____ entitled "Disaster Recovery with Bounded Data Loss" filed on May 10, 2001 (attorney docket no. 50277-1003, client docket no. 2000-207-01) by Mahesh Girkar et al., the contents of which are hereby incorporated by reference.

[39] Some database management systems provide a log viewer 129 to allow the user to extract data from recovery logs 117 for auditing or reporting. When used for this purpose, the recovery logs 117 constitute an audit trail of the changes that have been previously made to the OLTP database 113. In accordance with one embodiment of the invention, the shipped recovery logs 117 in the change source 131 are registered with the log viewer 129 to permit information to be extracted from the shipped recovery logs 117, not just for purposes of auditing as previously used but for change data capture. Use of a log viewer 129 advantageously encapsulates, hides, and insulates the implementation details of the recovery logs 117 from the publisher 127, thereby simplifying the design and maintenance of the publisher 127. Furthermore, the log viewer 129 permits the publisher 127 to extract data from recovery logs 117 that have been produced by a

foreign database system, which has been developed by a different database vendor and employs a different, incompatible implementation of the internal data structures, file formats, and algorithms.

[40] Preferably, the log viewer 129 is configured to provide a relational database interface to the data in the recovery logs 117 so that the publisher 127 can obtain the change data by executing a SQL select statement. For the purpose of the following discussion, the relational database interface is mediated through a relational view on the recovery logs 117 called `VIEWER`, which contains at least the following columns: `TABLE_VIEWED`, which contains the name of the source table from which change data is to be extracted, and SCN, which contains the system change number of each of the transactions in the recovery logs 117. A system change number is a monotonically increasing number used to unique identify operations performed on the OLTP database 113. Furthermore, a SQL function, `EXTRACT()`, is provided to extract data from a particular column from the recovery log 117. The `EXTRACT()` is passed the name of the column whose data is to be returned.

[41] At step 305, each change table is processed in a change set, which is a logical group of related change tables that are to be kept transactionally consistent. Accordingly, a transaction to update the change tables is begun for all the change tables in the change set and is committed at the end of this process to ensure that all the changes to related change table will become visible to the subscriber applications 121, 123 at the same time.

[42] At step 307, a structured query language (SQL) statement is generated to fetch the change data via the relational database interface provided by the log viewer 129 and insert the change data into a corresponding change table. In one implementation the generated SQL statement has the following form:

```
INSERT INTO ct (change_columns)
  SELECT source_columns FROM VIEWER
  WHERE TABLE_VIEWED=st AND SCN > ft AND SCN <= lt
```

[43] In this SQL statement, "`VIEWER`" is the name of the relational view that the log viewer 129 presents to the publisher as a relational interface; "`TABLE_VIEWED`" is a column that indicates which source table on the OLTP database 113 that has change data; and "`SCN`" is a

column that identifies the transaction that resulted in the change data. The remaining slots of the SQL statement, set in italics, are explained with respect to a working example as follows.

[44] The slot "*ct*" is the name of the current change table being processed in the loop controlled by step 305, and the slot "*st*" is the name of the corresponding source table. In the working example, if staging system 120 is configured to maintain a change table that corresponds to a source table in the OLTP database 113 called, the name of the corresponding change table may be SYSTEM.CT_EMP. In this example, the slot "*ct*" is filled with "SYSTEM.CT_EMP" and the slot "*st*" is filled with "SCOTT.EMP".

INS. A9

[45] The slot "*source_columns*" is a list of the columns in the change table In this example, if the SCOTT.EMP source table contains source columns EMPNO, ENAME, HIREDATE, and DEPTNO, then the "*source_columns*" slot would be filled with "EMPNO, ENAME, HIREDATE, DEPTNO". The string for this list may be built by iterating over the information stored in system metadata 230.

[46] The slot "*change_columns*" is a list of the columns in the change table and their corresponding source table columns with remaining as appropriate, with the general form "EXTRACT(*sc*) AS *cc*" where "*sc*" is the name of the source table column and the "*cc*" is the name of the change table column. In this example, if the change table columns are given the same names as the corresponding columns in the source table, then the "*change_columns*" slot would be filled with "EXTRACT(EMPNO) AS EMPNO, EXTRACT(ENAME) AS ENAME, EXTRACT(HIREDATE) AS HIREDATE, EXTRACT(DEPTNO) AS DEPTNO". The string for this list may be built by iterating over the columns in the change table, a list of which may be stored and obtained from system metadata 230.

[47] The slot "*ft*" holds the transaction identifier of the first change record for the current change table, in which the transaction identifier is either a unique, monotonically increasing transaction number or a system change number. Similarly, the slot "*lt*" is the transaction identifier of the last change record for the current change table. In this example, the "*ft*" slot may have a value of 1001 for the first transaction, and the "*lt*" slot may have a value of 9009 for the last transaction.

[48] The generated SQL statement, filling the slots with the data in accordance with this example, would be:

```
INSERT INTO SYSTEM.CT (EMPNO, ENAME, HIREDATE, DEPTNO)
  SELECT EXTRACT(EMPNO) AS EMPNO,
         EXTRACT(ENAME) AS ENAME,
         EXTRACT(HIREDATE) AS HIREDATE,
         EXTRACT(DEPTNO) AS DEPTNO
  FROM VIEWER
  WHERE TABLE_VIEWED=SCOTT.EMP AND SCN > 1001 AND SCN <= 9009
```

[49] At step 309, the generated SQL statement is executed, thereby fetching the change data from the shipped recovery logs 117 via the log viewer 129 and update the corresponding change tables. The SQL "`INSERT ... SELECT`" statement can be optimized by the analysis database 125 when its native language is SQL. As a result, the analysis database 125 able to take advantage of the indexing, partitioning, and parallel features of the analysis database 125. Furthermore, use of standard SQL for this operation benefits from the data integrity and transactional capability that is the hallmark of a modern relational database management system. For example, a plurality of the SQL "`INSERT ... SELECT`" statements can be made part of a single transaction per change set, to that the entire data loading operation executes as a single transaction. This approach addresses data integrity or recovery problems that characterize the conventional flat file approach that would arrive, for example, if columns EMPNO, ENAME, and HIREDATE successfully loaded, but column DEPTNO contained an error, impacting transactional consistency.

HARDWARE OVERVIEW

[50] FIG. 4 illustrates a computer system 400 upon which an embodiment according to the present invention can be implemented. The computer system 400 includes a bus 401 or other communication mechanism for communicating information, and a processor 403 coupled to the bus 401 for processing information. The computer system 400 also includes main memory 405, such as a random access memory (RAM) or other dynamic storage device, coupled to the bus 401 for storing information and instructions to be executed by the processor 403. Main memory 405 can also be used for storing temporary variables or other intermediate information during

execution of instructions to be executed by the processor 403. The computer system 400 further includes a read only memory (ROM) 407 or other static storage device coupled to the bus 401 for storing static information and instructions for the processor 403. A storage device 409, such as a magnetic disk or optical disk, is additionally coupled to the bus 401 for storing information and instructions.

[51] The computer system 400 may be coupled via the bus 401 to a display 411, such as a cathode ray tube (CRT), liquid crystal display, active matrix display, or plasma display, for displaying information to a computer user. An input device 413, such as a keyboard including alphanumeric and other keys, is coupled to the bus 401 for communicating information and command selections to the processor 403. Another type of user input device is cursor control 415, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to the processor 403 and for controlling cursor movement on the display 411.

[52] According to one embodiment of the invention, change data capture is provided by the computer system 400 in response to the processor 403 executing an arrangement of instructions contained in main memory 405. Such instructions can be read into main memory 405 from another computer-readable medium, such as the storage device 409. Execution of the arrangement of instructions contained in main memory 405 causes the processor 403 to perform the process steps described herein. One or more processors in a multi-processing arrangement may also be employed to execute the instructions contained in main memory 405. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the embodiment of the present invention. Thus, embodiments of the present invention are not limited to any specific combination of hardware circuitry and software.

[53] The computer system 400 also includes a communication interface 417 coupled to bus 401. The communication interface 417 provides a two-way data communication coupling to a network link 419 connected to a local network 421. For example, the communication interface 417 may be a digital subscriber line (DSL) card or modem, an integrated services digital network (ISDN) card, a cable modem, or a telephone modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 417

may be a local area network (LAN) card (e.g. for Ethernet™ or an Asynchronous Transfer Model (ATM) network) to provide a data communication connection to a compatible LAN. Wireless links can also be implemented. In any such implementation, communication interface 417 sends and receives electrical, electromagnetic, or optical signals that carry digital data streams representing various types of information. Further, the communication interface 417 can include peripheral interface devices, such as a Universal Serial Bus (USB) interface, a PCMCIA (Personal Computer Memory Card International Association) interface, etc.

[54] The network link 419 typically provides data communication through one or more networks to other data devices. For example, the network link 419 may provide a connection through local network 421 to a host computer 423, which has connectivity to a network 425 (e.g. a wide area network (WAN) or the global packet data communication network now commonly referred to as the "Internet") or to data equipment operated by service provider. The local network 421 and network 425 both use electrical, electromagnetic, or optical signals to convey information and instructions. The signals through the various networks and the signals on network link 419 and through communication interface 417, which communicate digital data with computer system 400, are exemplary forms of carrier waves bearing the information and instructions.

IN6. A1 [55] The computer system 400 can send messages and receive data, including program code, through the network(s), network link 419, and communication interface 417. In the Internet example, a server (not shown) might transmit requested code belonging an application program for implementing an embodiment of the present invention through the network 425, local network 421 and communication interface 417. The processor 404 may execute the transmitted code while being received and/or store the code in storage device 49, or other non-volatile storage for later execution. In this manner, computer system 400 may obtain application code in the form of a carrier wave.

[56] The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to the processor 404 for execution. Such a medium may take many forms, including but not limited to non-volatile media, volatile media, and transmission media. Non-volatile media include, for example, optical or magnetic disks, such as

storage device 409. Volatile media include dynamic memory, such as main memory 405. Transmission media include coaxial cables, copper wire and fiber optics, including the wires that comprise bus 401. Transmission media can also take the form of acoustic, optical, or electromagnetic waves, such as those generated during radio frequency (RF) and infrared (IR) data communications. Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, any other magnetic medium, a CD-ROM, CDRW, DVD, any other optical medium, punch cards, paper tape, optical mark sheets, any other physical medium with patterns of holes or other optically recognizable indicia, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave, or any other medium from which a computer can read.

[57] Various forms of computer-readable media may be involved in providing instructions to a processor for execution. For example, the instructions for carrying out at least part of the present invention may initially be borne on a magnetic disk of a remote computer. In such a scenario, the remote computer loads the instructions into main memory and sends the instructions over a telephone line using a modem. A modem of a local computer system receives the data on the telephone line and uses an infrared transmitter to convert the data to an infrared signal and transmit the infrared signal to a portable computing device, such as a personal digital assistance (PDA) and a laptop. An infrared detector on the portable computing device receives the information and instructions borne by the infrared signal and places the data on a bus. The bus conveys the data to main memory, from which a processor retrieves and executes the instructions. The instructions received by main memory may optionally be stored on storage device either before or after execution by processor.

CONCLUSION

[58] While the present invention has been described in connection with a number of embodiments and implementations, the present invention is not so limited but covers various obvious modifications and equivalent arrangements, which fall within the purview of the appended claims.